Lab 3
Interrupts and Visualization Tools

September 6, 2012

In this lab you will learn how to use hardware and software interrupts. Also, you will get familiar with the visualization tools and the real time data exchange capability of the C6713DSK.

1 Introduction

The DSP/BIOS real-time operating system available on the CCS provides real-time scheduling, analysis, and data transfer capabilities for an application running on the DSP. The DSP/BIOS has a preemptive real-time scheduler that determines which one of a number of different threads is executed by the DSP at any given time. Threads are DSP/BIOS objects that contain program code (functions). There are five different threads that can be used in a DSP/BIOS application:

1. Hardware interrupts (HWIs) have the highest priority. Their execution is triggered by interrupts from peripherals and they always run to completion.

2. Software interrupts (SWIs) are triggered from within a program. They run to completion unless preempted by a high priority SWI or HWI.

3. Periodic functions (PRDs) are a special type of SWI triggered by a dedicated hardware timer.

4. Tasks (TSKs) are created dynamically within a DSP/BIOS application, and they will start execution at the start of the DSP/BIOS application.

5. Idle functions (IDLs) are executed repeatedly as a part of the lowest priority thread. They contain functions that communicate real-time data analysis from the DSP to the Host.

The problem with hardware interrupts, is that the CPU cant service multiple interrupts at the same time. This will cause events to be missed and interrupts to be ignored and real-time may be missed. SWIs are triggered within programs; and different SWIs will be serviced based on the priority level which when designed correctly will help achieve real-time.

The C6713 comes with a variety of visualization tools. Using the DSP/BIOS real-time analysis tools the performance of the DSP may be monitored, and hence provide a tool for optimization. Some of these tools include:

- CPU load graph
- Execution graph
- Message log

In order to observe the operation of the DSP, `printf` may be used to display the values of certain variables. This will work, but the execution of the `printf` function will load the CPU and eventually will prevent it from achieving real-time. Another way is to create a log-event and use the `LOG printf` function instead. The `LOG` object inserted sets up a buffer in which the `LOG printf` function can append messages. The buffer contents is sent to a Host computer in real time during the execution of an Idle function.

2 Lab

2.1 Part 1: Hardware Interrupts

Create a program to receive a signal at the LINE IN input and retransmit it through the LINE OUT output in real-time. (What does 'real-time' mean ?)

1. Start a new project
2. Create a DSP/BIOS file and add it to the project
3. Create a hardware interrupt that is triggered by the data received from the MCBSP1. In the hardware service routine, read the data and retransmit it. Add to your source file `#include csl irq.h`. The CSL helps creating, configuring and using the interrupts.
   (a) Open the DSP/BIOS FILE and click on `Scheduling`.
   (b) Click on `HWI-Hardware Interrupt Service Routine Manager`.
   (c) Select `HWI_INT11` by right-clicking, then select `Properties`.
   (d) In the interrupt source field select `MCSP_1_Receive`.
   (e) In the `function field` enter the name of the hardware interrupt function that you want to create. **This field needs the assembly name of the function which is the name that you use in C preceded by an underscore**.
   (f) Click on the `Dispatcher` tab and select `Use Dispatcher`.
4. Add the following statements to enable interrupts after you initialize the board and the codec:

```
IRQ_map(IRQ_EVT_RINT1,11); // Map McBSP1 using CSL function
IRQ_clear(IRQ_EVT_RINT1);
IRQ_globalEnable();        // Enable ints globally with bios function
IRQ_enable(IRQ_EVT_RINT1); // Enable McBSP1 using CSL function
```

5. Test your hardware interrupt by connecting the input to either the function generator or to any other source and make sure that you are getting the same thing out.

2.2 Part 2: Software interrupts

Now we will try to create a software interrupt that will take advantage of the scheduling capability of the board. Here is how you can create a software interrupt.

1. Open the DSP/BIOS FILE and click on `Scheduling`
2. Right-Click on `SWI-Software Interrupt Manager` and select `Insert SWI`.
3. Rename the `SWI` to something meaningful.
4. View the properties of the created `SWI` and insert the name of the function that you intend to create. **Again do not forget to precede the name of the software interrupt function by an underscore**.
5. In your source file create the software interrupt function that will be executed when an interrupt occurs.
6. You can `cut` and `paste` the same code that you had in your hardware interrupt function (`HWI`).
7. Now in your `HWI` interrupt function use the following function: `SWI_post(&swi_handle)`. Where `swi_handle` is the name of your software interrupt.
8. Test your code and make sure it works.

2.3 Part 3: CPU load

1. Turn on the `CPU Load Graph` from the `DSP/BIOS` menu, run your code and record the CPU load.
2. Add to your source file: `#include <stdio.h>`, and modify your program to use the `printf` function to display the data that you are reading.
3. What is the CPU load ? Explain.
4. Now we will create a *log event*.
   (a) Open the `DSP/BIOS FILE` and click on `Instrumentation`.
   (b) Right-Click on `LOG - Event Manager` and select `Insert LOG`.
   (c) Rename the log event to something meaningfull.
   (d) View the properties of the log event you just created and select `printf` to be the `datatype`.
   (e) Now, in your code, instead of using the `printf` use `LOG_printf`. *It is your responsibility to find out the required arguments for this function*.
   (f) Turn on the `Message Log` from the `DSP/BIOS` menu.
   (g) Run your code and make sure it works.
5. What is the CPU load?

6. Now try to find the CPU load difference between configuring the MCBSP to read 16 bits from one channel and then do another 16-bit read to get the other channel, and configuring the MCBSP to read 32 bits at once and then use logic and bit operations to separate the two channels.
   (a) Set the sampling rate to 8kHz
   (b) Monitor the CPU load while performing a 32-bit read from the codec.
   (c) Monitor the CPU load with the default setting that allows you to read only 16 bits at a time.
   (d) Which approach is faster and why?